Solving the Permutation Flow Shop Problem with Makespan Criterion using Grids

Samia kouki[1], Mohamed Jemni[1], Talel Ladhari[2, 3]

[1] *Ecole Supérieure des Sciences et Techniques de Tunis, Research Laboratory UTIC. University of Tunis, Tunisia.*
[2] *ROI – Combinatorial Optimization Research Group, Polytechnic School of Tunisia, B.P. 743, 2078, La Marsa, Tunisia*
[3]*Princess Fatimah Al-Nijriss Research Chair for Advanced Manufacturing Technologies, Department of Industrial Engineering, College of Engineering, King Saud University, PO Box 800 Riyadh 11421, Saudi Arabia.*

Samia.kouki@yahoo.fr, Mohamed.jemni@fst.rnu.tn, talel_ladhari2004@yahoo.fr

Abstract

The optimization of scheduling problems is based on different criteria to optimize. One of the most important criteria is the minimization of completion time of the last task on the last machine called makespan. In this paper, we present a parallel algorithm for solving the permutation flow shop problem. Our algorithm is a basic parallel distributed algorithm deployed in a grid of computer (Grid'5000). The objective of this work is minimizing the total makespan of the tasks. Our algorithm uses the exact Branch and Bound method to find optimal solutions of the problem through the distribution of the tasks among the available processors. Computational results of our parallel algorithm using well known Taillard's benchmarks, showed encouraging results. In particular, we succeeded to solve two new instances to optimality which had never been resolved before neither in sequential nor in parallel [29].

Keywords: *Permutation Flow Shop problem, Branch and Bound algorithm, parallel algorithms, Parallel computing, grid computing.*

1. Introduction

Today we are witnessing an explosion in the amount of information and data used on many fields as industry, marketing, research, etc… The management of these huge data needs optimization tools and techniques. That is why scientists have focused, since several years, on improving these techniques of optimization, in particular, in the industrial areas. In this context, the objective of our work is the optimization of scheduling jobs problem in the textile industry. Our goal is to minimize the completion time of the last job on the last machine. The interest in this classical NP-hard problem is motivated by its practical relevance and its challenging hardness. Specifically, we are interested on the Permutation Flow Shop Problem (PFSP) which is one of hard problems in combinatorial optimization and scheduling theory. In fact, there exist two different classes of methods to solve such kind of problems: approximate and exact methods. In this paper, we are interested in the exact method based in the Branch and Bound (B&B) technique to find the optimal solution of the problem and confirms its optimality. Nevertheless, the resolution of some problem instances by exact methods may require weeks, months or years of computation. Specifically for large scale instances, an exact resolution is often impractical due to the limited amount of resources of classical machines and computing environments. This kind of problems needs a computing environment that delivers large amounts of computational power over a long period of time. Precisely, the proliferation of high performance systems and the emergence of high speed networks (terabit networks), like as parallel and grid computing, can be good alternatives to reduce the computation time of such problems.

In this context, we present, in this paper, a parallel distributed algorithm to solve the PFSP, based on the B&B method, in the grid. Our parallel algorithm distributes the whole search tree among all processors and applies the B&B algorithm in the local portion of data at each processor. This algorithm treats well known lower Bound (***LB7***) [2], and has been tested with Taillard's benchmarks [1], using the French grid "Grid'5000" [27].

Thanks to the huge number of resources provided by the grid computing, our objective is to propose an efficient algorithm in order to solve the PFSP and in particular some unsolved instances of Taillard [1]. Reducing the running times of instances requiring many hours and even years to be solved is another object of our work. The originality of our work is the use of the so-called ***LB7*** lower bound [2] to solve the PFSP. This lower bound is composed of tree levels; in every level a different lower bound is computed.

This paper is organized as follows: in the next section, we present the sequential B&B algorithm for the PFSP. In section 3, we present the state of the art related to parallel B&B algorithm. Section 4 is dedicated to our Gridified Algorithm *Updating the Upper Bound* (GAUUB), for the PFSP. In section 5, we present our computational results performed on well-known benchmarks (Taillard), using the GAUUB algorithm. We conclude the paper by a conclusion and the presentation of our future perspectives.

2. The Permutation Flow Shop Problem (PFSP)

2.1. Presentation

The theory of scheduling is characterized by a virtually unlimited number of problem types [3]. Generally, in a scheduling , we suppose we have $\boldsymbol{m}$ machines to process $\boldsymbol{n}$ jobs. A schedule is assigning, for each job, an allocation of one or more time intervals to one or more machines. Schedules may be represented by Gantt charts [3].

Thus, the flow shop problem (FSP) is one of the most important problems in the scheduling theory. However, most of research has focused on the flow shop (PFSP), which can be considered as a classical flow shop problem by adding the assumption: the jobs must be processed in the same sequence by each of the $\boldsymbol{m}$ machines.

This problem can be described as follows. Each job $\boldsymbol{j_i}$ ($\boldsymbol{i=1, 2\dots n}$) has to be processed on $\boldsymbol{m}$ machines $\boldsymbol{M_j}$ ($\boldsymbol{j = 1, \dots, m}$), following the same order in all machines. The processing time of job $\boldsymbol{j_i}$ on machine $\boldsymbol{M_j}$ is $\boldsymbol{p_{ij}}$. To solve such a problem we have to consider many constraints as:

- All jobs are ready for processing at time zero.
- The machines are continuously available from time zero onwards (no breakdowns).
- At any time, each machine can process at most one job and each job can be processed on at most one machine.
- No pre-emption is allowed (that is, once the processing of a job on a machine has started, it must be completed without interruption).

- Only permutation schedules are allowed (i.e. all jobs have the same ordering sequence on all machines).

Not only constraints are considered in a PFSP, but we have also an objective function to satisfy. In this work, we focus on the minimization of the completion time of the last job of the last machine called makespan and denoted C_{max}. Suppose that we have a discrete set S, and an optimal solution $x^* \in S$, such that $F(x^*) = \min F(x)\ \forall x \in S$, where F is called the objective function, S is called feasible region or domain and x feasible solution. Thus, solving the PFSP having the makespan as objective function to be minimized consists on determining the permutation of jobs which gives the smallest makespan value. This problem is denoted $F|prmu|C_{max}$. Figure 1 shows an example for the execution of three jobs on two machines.

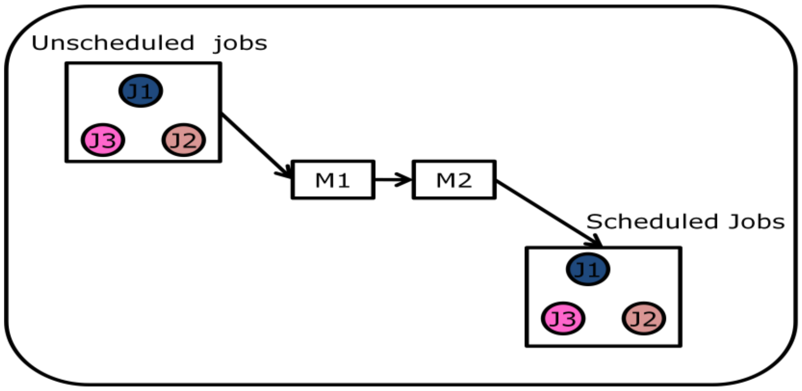
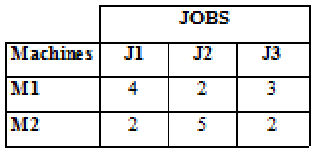


Figure 1-An Example of the PFSP

2.2. Example of the PFSP

In order to illustrate the PFSP, we consider the following simple example: let there be three jobs each of which to be processed on two machines ***M1***and ***M2*** in the order ***M1*** then ***M2***. The workflow diagram of this problem is shown in Figure 2. The processing time matrix (in minutes) is given in Table 1. For the above three jobs and two machines PFSP, the schedule of the jobs can be either ***J1–J2–J3*** or ***J1–J3 –J2*** or ***J2– J1–J3*** or ***J2–J3–J1*** or ***J3–J1–J2*** or ***J3–J2 –J1*** (3!). Figure 3, shows the Gantt chart presenting our example.



	JOBS		
Machines	J1	J2	J3
M1	4	2	3
M2	2	5	2

Table 1. Processing time matrix

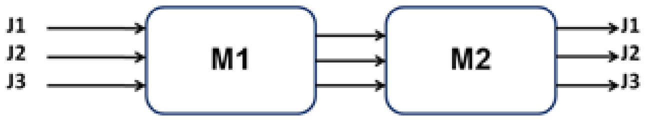


Figure 2: Workflow in PFSP

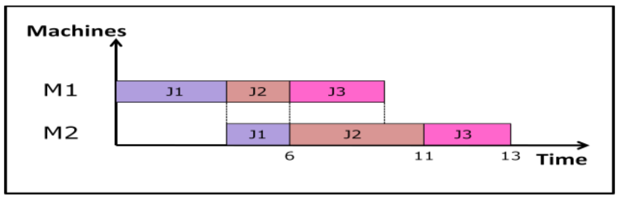


Figure 3-Gantt Chart for the Schedule J1–J2–J3

2.3. Complexity

Practical experience shows that some computational problems are easier to solve than others. Complexity theory provides a mathematical framework in which computational problems are studied so that they can be classified as "easy" or "hard" [3].

For the pure flow shop problem, there are generally (***n!***)****m*** different alternatives for sequencing jobs on machines. However, for the PFSP, since it considers the same order of processing all the jobs in all machines, the search space is reduced to ***n!***.
Consequentially, the ***n***-job ***m***-machine PFSPs belong to the class of NP-hard problems [4]. Thus, in a PFSP, the computational requirements for obtaining an optimal solution increase exponentially as problem size increases. Nevertheless, it is well-known that the case of the PFSP composed of two machines ($F2 \mid \mid C_{max}$), could be easily solved using Johnson's rule which generates an optimal schedule in O(n*log(n)) time [5][6]. However, for $m \geq 3$, the problem is shown to be strongly NP-hard [7].

2.4 Solving the PFSP by the Branch & Bound Method

There exists much kind of methods to solve combinatorial optimization problems depending on the complexity of the problem to solve. As mentioned above, there exist mainly two classes of methods; exact methods and approximate methods. Exact methods are limited by the size of the problem to solve, as they become impractical for large size problems. While, the approximate methods (heuristics and meta-heuristics), give good results even though the size of the problem increases. It should be noted that approximate methods do not guarantee the optimality of solutions found.

In this work, we are interested on finding the minimum completion time of the last job on the last machine called the makespan, of the PFSP, using the exact B&B method.
The Branch and Bound method is based on the idea of intelligently enumerating all feasible solutions. Indeed, the B&B algorithm is an exact method applied in order to obtain the optimal solution to a wide class of combinatorial optimization and scheduling problems. It was introduced by Ignall and Shrage [8].

The principle of the B&B method is to make an implicit search through the space of all possible feasible solutions of the problem (represented by a search tree). These enumerative optimization methods require a substantial computing time and can, subsequently, solve only relatively small sized instance problems. In this subsection, we describe briefly the basic components of a classical B&B algorithm. The B&B method must include the three following basic components:

- A branching strategy that splits the region of feasible solutions into sub-regions or sub-problems through the addition of constraints. Convergence of B&B is assured if the size of each new generated sub-problem is smaller than the original problem, and the number of feasible solutions to the original problem is finite.
- The lower bounding is an algorithm which provides a lower bound (***LB***) for each sub-problem generated by the branching scheme. As more as the bound is stronger, as more it eliminates nodes from the search tree. But if its computational requirements turn excessively large, it may become advantageous to search through larger parts of the tree, using a weaker but more quickly computable bound (a weak bound and a large space).
- A search or an exploration strategy selects the node from which to branch. For instance, there is the depth first strategy, the large first strategy and the best first strategy.

Figure 4 presents an example of a B&B search tree, solving the PFSP. In this example, we have to schedule five jobs having as initial Upper Bound (UB) equal to 40, we compute the value of the Lower Bound in each node of the tree. Once we reach a leaf node (feasible solution), we update the Upper Bound value, and we continue with the same manner until exploring all potential branches of the search tree. When the whole search tree is explored, we confirm that the optimal solution is found.

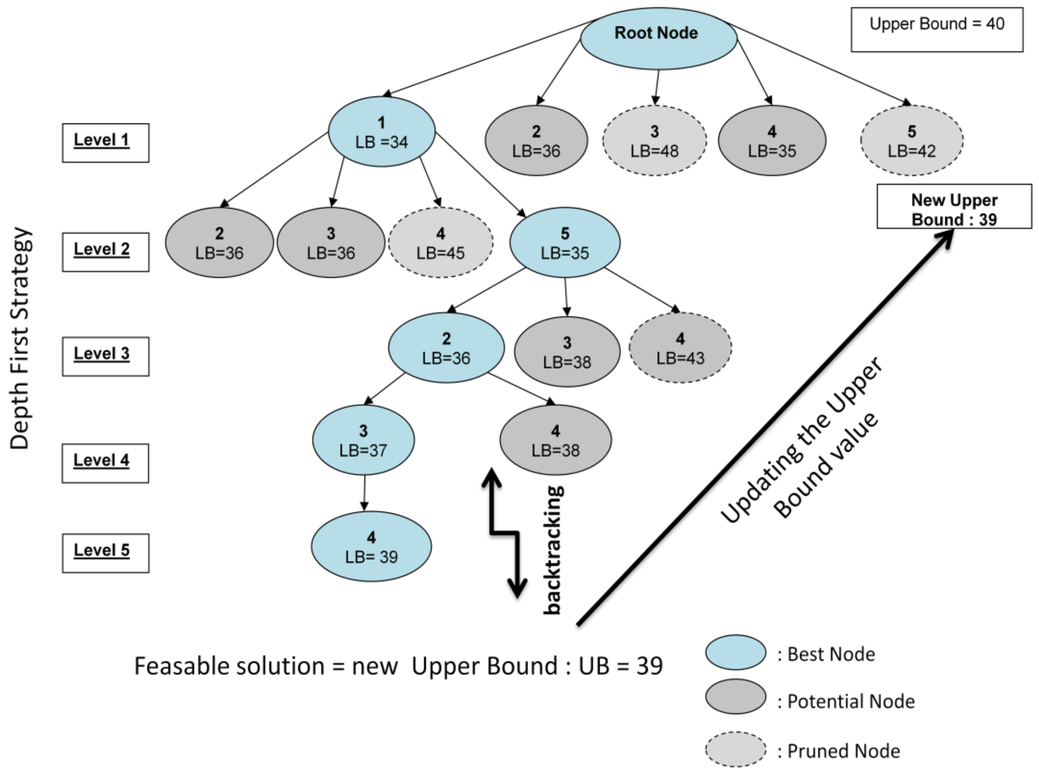


Figure 4-Sequential B&B Algorithm for the PFSP

3. Parallel B&B Algorithms

3.1. Introduction

The objective of a parallel system is to make as faster as possible the execution of applications (faster than they can be executed on single processor systems). For this aim, the application should be restructured and written in a form that allows it to benefit from the multiple processing units. The use of parallel machines improves the time needed to reach optimal solutions as well as it makes possible resolving problems with very big sizes. Several factors are combined to use parallel computing to solve the combinatorial optimization problems. Indeed, finding an optimal solution for a problem can be very difficult and sometimes impossible on a single processor. Many previous works dealt with parallel B&B as reported in [9] for many reasons. A first one is the scientific progress in Operations Research, in particular, regarding the quality of the lower bounds for these problems. However, the computation of these new bounds is very time consuming. Moreover, the bounds are computed at each node of a tree with a huge size (can reach several billions of nodes). The use of parallel branch and bound (B&B) strategies on large computer clusters and grids with advanced programming tools, including multithreading and fault tolerance functionalities, is the second factor of success. The progress in processor computing power certainly constituted another factor.

3.2. State of the Art

The parallelization of the B&B algorithm has been widely studied in the last two decades. Many previous studies were interested in the parallelization of the B&B algorithm. For instance, an exhaustive survey of the state of the art of parallel B&B algorithms, covering the period from 1975 to 1994, has been presented by B. Gendron and T. Crainic in [10].

After this period, in 1995, S. Okamoto, I. Wantanabe and H. Lizuka, presented in [11] a parallel B&B algorithm for the flow shop problem and implemented in a parallel machine nCUBE2 multiprocessor which is a shared memory parallel machine. After that, in 2003, K. Aida and al. proposed in [12] a parallel algorithm dedicated especially for the resolution of the BMI Eigenvalue Problem with the hierarchical master-worker paradigm. In 2004, the algorithm proposed in [13], presented another algorithm devoted to shared memory parallel machines. However, the authors, D. A. Bader did not present experimental results dealing with the PFSP but a general parallel B&B. D. Caromel and al., in [14] presented a framework for using B&B algorithm in grid computers. They used in this framework a hierarchical master-worker strategy and used the flow shop problem to test their framework. Remark that, the authors used randomly generated instances and did not used known benchmarks.

Moreover, many other frameworks for parallel B&B algorithms have been designed to solve some combinatorial optimization problems; we present some examples of them.
PUBB [15], is a framework which proposes a C interface and allows parallelizing B&B algorithms on any specific combinatorial optimization problem. Bob++ library is a set of C++ classes. Its goal is to allow the implementation of sequential and parallel search algorithms (Branch and X, Dynamic programming, etc) to solve a specific combinatorial optimization problem. In [16], the authors presented many others frameworks dealing with parallel B&B method in shared and distributed environments, like PPBB-Lib, MallBa, ZRAM. These frameworks used a C interface. However, others frameworks as ALPS/BiCePS, PICO [17], and MW proposed a C++ interface.

In [18], authors propose a P2P design and implementation of a parallel B&B algorithm on the ProActive grid middleware. They applied their application to the flow shop scheduling problem and experimented on a computational pool of 1500 CPUs from the GRID'5000 Nation-wide experimental Grid. Remark that the authors used other benchmarks than we used in our study. In [19], Benjamin W. Wah and Y. W. Ma proposed a parallel machine for processing nondeterministic polynomial complete problems, and evaluate their system using the vertex covering problem. In [20], J. Lemesre, C. Dhaenens and E.G. Talbi propose an exact parallel B&B algorithm, using the bi-objective PFSP where the optimal solutions are found using the Pareto approach and represent a set of solutions of best compromise. In [21], M. Mezmaz, N. Melab and E.-G. Talbi present a parallel B&B algorithm using grid'5000, they resolved the tail056 Taillard's benchmark in 25 days. In fact, although this may be considered as a promising result, more investments have to be investigated to reduce the running time which is still huge.

Unlike the work cited above, in this paper we deal specifically with the PFSP. To our knowledge, the parallelization of this particular problem has not been studied in its generality, and just few works have been achieved to solve particular instances. In this context, we propose a new parallel distributed algorithm for PFSP with makespan criterion. We are interested, in particular, to the well known benchmarks of Taillard [1], presenting some hard instances not yet solved, and this constitutes the originality of our work. We highline here, that some instances of the PFSP [22] are still not yet solved neither with sequential algorithms nor with parallel ones.

Since the publication of the seminal paper of Johnson (1954) [23], the PFSP has became one of the most intensively investigated topics in combinatorial optimization and scheduling theory. This interest is not only motivated by its practical relevance, but also by its deceptive simplicity and challenging hardness. Nevertheless, the flow shop problem is still considered as a very hard nut to crack. Indeed, up to the mid of the 1990s, the best available branch-and-bound algorithms experience difficulty in solving instances with 15 jobs and 4 machines [24]. In this context, ETSI and INRIA at Sophia Antipolis France organized in 2005 the 2nd Grid PLUGTESTS Flow Shop Challenge Contest [25]. The goal of this contest is focused on solving the well known benchmarks problems of Taillard [1]. Indeed, in 2007, a challenge was launched in order to solve some of the instances of combinatorial optimization problem, for instance the PFSP, and a call for competition has been announced in the web [26].

4. Our Gridified Algorithm Updating the Upper Bound (GAUUB)

The high computing time of the PFSP solved by B&B algorithm is due to the computation of bounds calculated at each node of the search tree (as presented in Figure 4). The size of the B&B tree can ne huge and can attempt several billions of nodes for large sizes of problems. Furthermore, not all combinatorial optimization problems can be parallelized, since this depends strictly on the nature and the characteristics of the problem to treat, and in particular on the quality of the upper and lower bounds.

Our implemented algorithm is based on updating the upper bound value in order to make the algorithm converging rapidly to the optimal solution. Also, the strategy of exploring the search tree is very important for the resolution of hard problems. In this algorithm (GAUUB), we use the known lower bound (***LB7***) [2] which minimizes the size of the search tree by pruning more branches. Our experimentations are realized by the use the French grid Grid’5000 [27].

Grid'5000 is a French research grid offering a large scale nationwide infrastructure for large scale parallel and distributed computing research. Seventeen laboratories are involved in France with the objective of providing the community a test bed allowing experiments in all the software layers between the network protocols up to the applications [27]. Figure 5 shows clearly the architecture of the Grid’5000 and its principal components.

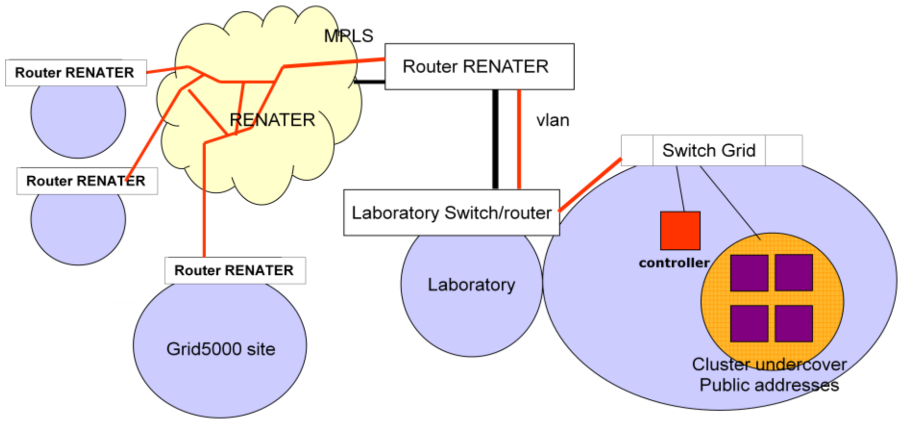


Figure 5- The Grid’5000 Architecture

Our parallelization strategy is based on dividing the search tree to many sub-problems to be performed independently and simultaneously by different processors. However, the parallel execution of the B&B solving the PFSP, can lead to the treatment of some needless sub-problems, although some of these problems may be pruned in sequential execution. For this reason, in our algorithm, we used an intelligent technique to distribute and update the upper bound values in order to reduce the size of the search tree and therefore to obtain as quickly as possible the optimal solution.

In this parallel algorithm, we have to distribute all the jobs to be scheduled among all processors. This task is done by the master. Then, each processor is responsible of its local sub-problem (set of jobs). Each processor has to use the operating times of different jobs on different machines in order to build its complete local sub tree.

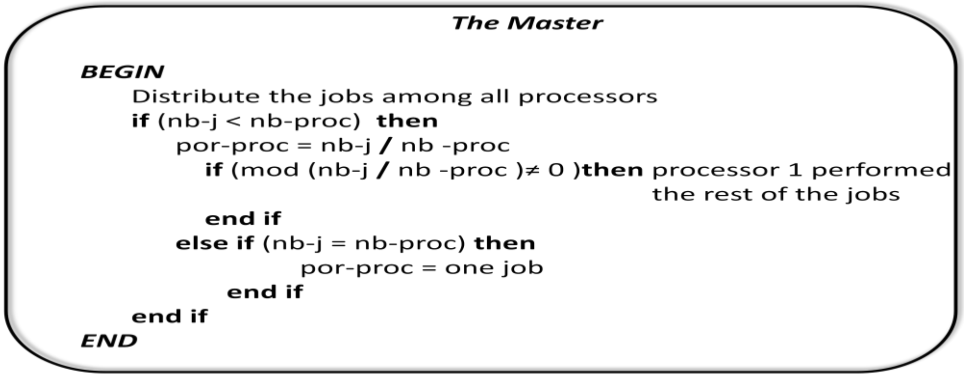
While developing the nodes of its sub tree, when the processor reach a feasible solution (which has a lower bound value less than the upper bound value), it updates the value of the Upper bound (if ***LB*** < ***UB***) and sends this new value to all other processors in order to prune the maximum of local not promising branches. We need then communication between all processors in this step of the algorithm.

Remark that for the parallel execution of the algorithm, the structure of the search tree may be different from the tree developed in sequential, since it may explore some branches of the tree which are not developed in the sequential version. However, the time investigated in the development of unexplored branches may be awarded by a fast pruning of non promising branches.

Since we have a fairly high number of processors, we can decompose our problem into a number of smaller sub-problems equal to the number of processors. In this algorithm we use especially the update of the current best upper bound denoted by (***NUB***: New Upper Bound). We use the paradigm master/slave in order to distribute and compute upper and lower bounds for each portion of the tree.

The master has to distribute the input data (root node) among all the processors. It has the responsibility of controlling all the system. However, the *Worker* processors concurrently execute independent tasks.

We present below different steps of our algorithm (GAUUB), for the master and the workers processors.



```
                         The Master

BEGIN
    Distribute the jobs among all processors
    if (nb-j < nb-proc)  then
        por-proc = nb-j / nb -proc
           if (mod (nb-j / nb -proc )≠ 0 )then processor 1 performed
                                                the rest of the jobs
           end if
        else if (nb-j = nb-proc) then
                  por-proc = one job
             end if
    end if
END
```

Where:

nb-j is the number of jobs
nb-pro is the number of processors
por-proc is the number of jobs for each processor.

For the workers the treatment is different from the master.

```
                              Workers

BEGIN
    - Explore the node
    repeat
        ▪ Branch the child of the node
        ▪ Compute the lower bound
        ▪ Choose  the minimum (lower bound)
    until a leaf node
    - Construct a feasible solution :
    A New LB(feasible solution) = NLB = candidate to a NUB
    - Compare the LB to the NUB:
    if (NLB < NUB) then update the NUB and broadcast it
            to all the processors (all to all communication).
            else return to the step 1
    end if
END
```

Where:

NLB is the New Lower Bound

NUB is the New Upper Bound.

In order to avoid exploring some non promoting branches of the search tree, once a processor reach a leaf node have to compare its lower bound and the current upper bound. If the lower bound is less than the Upper bound then this new value is broadcasted is sanded to the master.

After that the master has to, first, to update the value of the Upper bound and second, to broadcast this new value to all processors. Figure 6, shows clearly how our algorithm works.

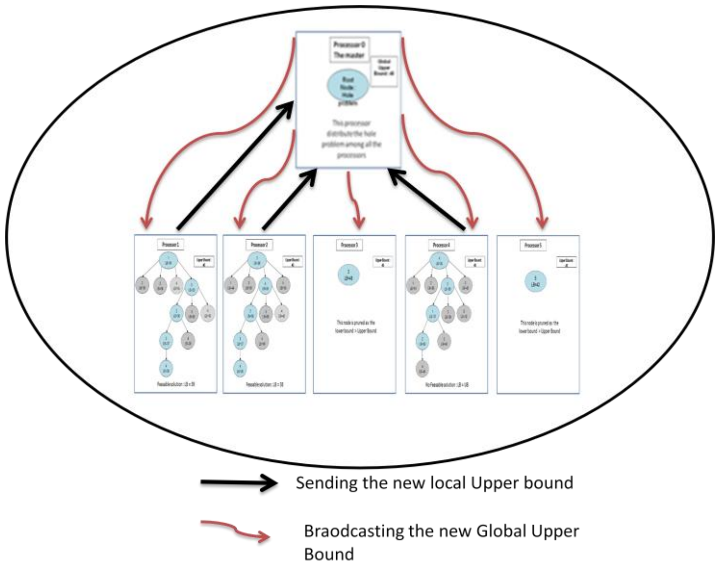

Processor 0
The master
Root Node Hole problem
This processor distribute the hole problem among all the processors
Processor 1
Processor 2
Processor 3
Processor 4
Processor 5
Sending the new local Upper bound
Braodcasting the new Global Upper Bound

Figure 6- Parallel Resolution of the PFSP with GAUUB

In the figure 6, we present an example of five processors solving the PFSP, with the GAUUB algorithm. All processors explore the search tree until reaching a leaf node. All processors send the value of the C_{max} of the new feasible solution to the master. The master compares all received values of C_{max} to the upper bound and updates it with a new value (if this value is less than the upper bound).

5. Computational Study

We implemented our algorithm with C and compiled it with Microsoft Visual C++ 2010. We used MPI library (Message Passing Interface) in order to ensure communication between processors. Thus, the most compelling reason for using a message passing model is its performance.

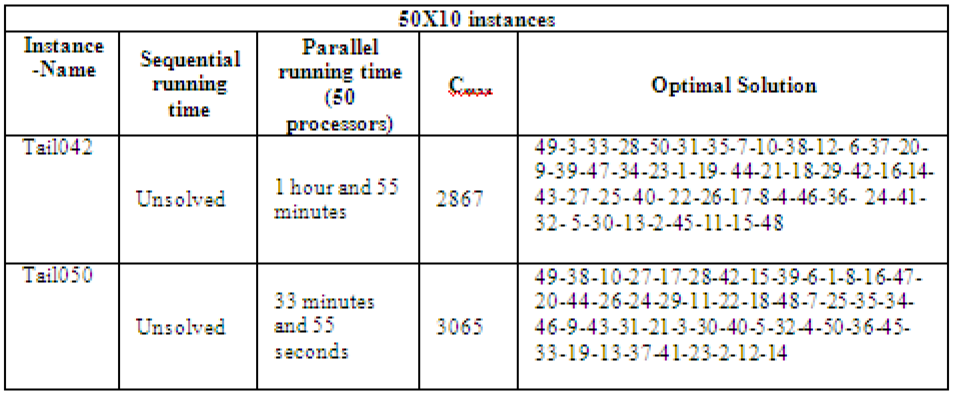
All the computational results were carried on the Grid'5000.

In order to validate practically the efficiency of our gridified algorithm we performed our experimental tests on a well-known set of benchmark instances from the literature [1]. These instances are known to be very hard and about a decade after their publication, many of them are still open (instances with 20 machines and some others with 10 machines).

In our experimentations we tested our application with the following instances of Taillard: 50X10, 100X10 and 200X10 instances.

We focused particularly in the 50X10 instances and we successfully solved two instances to optimality which had never been resolved before neither with sequential programs nor with parallel ones.

Table 2. Resolution of the 50X10 Instances in the Grid



50X10 instances				
Instance -Name	Sequential running time	Parallel running time (50 processors)	C_{max}	Optimal Solution
Tail042	Unsolved	1 hour and 55 minutes	2867	49-3-33-28-50-31-35-7-10-38-12- 6-37-20-9-39-47-34-23-1-19- 44-21-18-29-42-16-14-43-27-25-40- 22-26-17-8-4-46-36- 24-41-32- 5-30-13-2-45-11-15-48
Tail050	Unsolved	33 minutes and 55 seconds	3065	49-38-10-27-17-28-42-15-39-6-1-8-16-47-20-44-26-24-29-11-22-18-48-7-25-35-34-46-9-43-31-21-3-30-40-5-32-4-50-36-45-33-19-13-37-41-23-2-12-14

Remark here, that the resolution of the PFSP depends also on the instance to solve. In fact, from the experiments we have done, we noticed that when several nodes are pruned from the first level of the search tree, the use of parallelism is not very significant. However, when the processors share the workload that is normally assigned to a single processor, we can achieve reasonable running times for some instances.

During our experiments we have varied the number of processors starting with 1 CPU, 5, 10, 20, 50 and 100, depending on the size of the data instance to solve. Indeed, the results we obtained were dependent even in the instance within the same data group member size. For example, in some cases of data instances, we noticed during our experiments that our algorithm provides good accelerations and its deployment on the grid gives interesting execution times.

6. Conclusion and Perspectives

This paper presents a parallel algorithm for solving the PFSP as well as its deployment on a grid of computers (Grid'5000) and represents a continuation of our earlier works [30], [31], and [32] . Our algorithm is basically a parallelization of the well known B&B method, which is an exact method for solving combinatorial optimization problems. We presented, in particular, a new strategy of parallelization which uses some directives of communication between all processors in order to update the value of the Upper Bound. The deployment of our application on Gird'5000 gave good results and allowed in particular the resolution of two new instances not yet resolved, which are the Tail043 and Tail050 from Taillard's benchmarks [29]. In our future work, in order to more improve our algorithm and results, we plan to use a bigger number of processors, to solve other instances not yet solved and to use other paradigms, such us, the multithreading paradigm in order to more ameliorate load balancing between all processors. We noted in this paper a loud imbalance between the processors which affects the efficiency of our algorithm. Thus, we intend to establish a new strategy of load balancing between all processors.

References

[1] E. Taillard, "Benchmarks for basic scheduling problems", European Journal of Operational Research, 1993, vol. 64, pp. 278—285.

[2] T. Ladhari, and M. Haouari, "A computational study of the PFSP based on a tight lower bound", Computers & Operations Research, vol. 32 , 2005, pp. 1831--1847.

[3] P. Brucker, Scheduling algorithms, 2nd edition, Springer, Heidelberg, 1998.

[4] J. K. Lenstra, AHG. Rinnooy Kan, and P. Bruker, "Complexity of machine scheduling problems", Annals of Discrete Mathematics, 1977, vol. 1 , pp. 343-362.

[5] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included", Naval Research Logistics Quarterly, 1954, vol. 8, pp. 1-61.

[6] J. Carlier, I. Rebai, "Two branch-and-bound algorithms for the permutation flowshop problem", European Journal of Operational Research, 1996, vol. 90, n° 2, pp. 238--251.

[7] Garey, M. R., Johnson, DS., Sethi, R.: The complexity of flow shop and job shop scheduling. Mathematics of Operations Research, vol. 29, pp. 1:117, (1976).

[8] Ignall, E., Schrage, L. E. : Application of the branch-and-bound technique to some flow shop problems. Operations Research, vol. 12, pp. 400--412, (1965).

[9] Crainic, T. G. : Parallel Branch-and-Branch Algorithms: Survey and synthesis. OPERATIONS RESEARCH, Vol. 42, No. 6, pp. 1042--1066, Nov.-Dec. (1994).

[10] Gendron, B., Crainic, T. G. : Parallel B&B Algorithms: Survey and synthesis. Operation Research, Vol. 42, No. 6, pp. 1042--1066, November-December (1994).

[11] Okamoto, S., Wantanabe, I., Lizuka, H. :A new Par. algorithm for the n-job, m-machine flow-shop scheduling problem. Systems and Computers in Japan, Vol. 26, N°. 2, 1995.

[12] Aida, K. , Natsume, W., Futakata, Y. : Distributed computing with hierarchical master-worker paradigm for parallel B&B algorithm. CCGrid 2003,3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp.156--163, (2003).

[13] Bader, D. A. : Parallel algorithm design for branch and bound. International Series in Operations Research & Management Science, Vol. 76, pp. 5-1-5-44, (2004).

[14] Caromel, D., Di Costanzo, A., Baduel, L., Matsuoka, S. : Grid'BnB: A parallel B&B Framework for Grids. International conference on high performance computing (HIPC), (2007).

[15] Wang, X., Cheng, T. C. E., :Two-machine flowshop scheduling with job class setups to minimize total flowtime, Computers and Operations Research, Vol. 32 , Issue 11, pp. 2751 – 2770 Nov (2005).

[16] Crainic, T. G. : Parallel Branch-and-Branch Algorithms: Survey and synthesis. OPERATIONS RESEARCH, Vol. 42, No. 6, pp. 1042--1066, Nov.-Dec. (1994).

[17]Phillips, C. A., Eckstein, J., Hart, W.E. : Pico: An object oriented framework for parallel branch-andbound. Technical report, RUTCOR Research Report, (2000).

[18]Bendjoudi, A., Melab, N., Talbi, E.-G : P2P design and implementation of a parallel B&B algorithm for grids. International Journal of Grid and Utility Computing, vol. 1, pp. 159--168, (2009).

[19]Benjamin, W. Wah, Ma, Y. W. : MANIP-a parallel computer system for implementing B&B algorithm. International Symposium on Computer Architecture, pp. 239--262, (1981).

[20]Lemesre, J., Dhaenens, C., Talbi, E. G. :An exact parallel method for a bi-objective permutation flowshopproblem. European Journal of Operational Research, Volume 177, Issue 3, pp. 1641--1655, March (2007).

[21]Mezmaz M., Melab N., Talbi, E.-G. : A Grid-enabled B&B Algorithm for Solving Challenging Combinatorial Optimization Problems. Parallel and Distributed Processing Symposium, IEEE International, pp. 1-- 9, March (2007).

[22] Bellman, R., Esogbue, AO., Nabeshima, I., :Mathematical Aspects of scheduling and Applications. Pergamon Press", p 202, (1982).

[23]Johnson, S. M. :Optimal two- and three-stage production schedules with setup times included," Naval Research Logistics Quarterly, vol. 8, pp. 1-61, 1954.

[24] Anderson, E. J., Glass, C. A., Potts, C. N. :Local search in combinatorial optimization: Machine Scheduling. *Local Search in Combinatorial Optimization*, John Wiley and Sons, pp. 361--414, Chichester, United States.

[25] www-sop.inria.fr/oasis/plugtest2005/2ndGridPlugtestsReport/

[26] www2.lifl.fr/~talbi/challenge2007/.

[27] www.grid5000.fr/

[28] www.lri.fr/~fci/Grid5000/downloads/Compte-rendu.ppt

[29] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt

[30] Samia Kouki, Talel Ladhari, Mohamed Jemni, A Parallel Distributed Algorithm for the Permutation Flow Shop Scheduling Problem. ICA3PP (2) 2010: 328-337.

[31] Samia Kouki, Mohamed Jemni, Talel Ladhari, Design of parallel distributed algorithm for the Permutation Flow Shop Problem, NOTERE 2010: 65-72.

[32] Samia Kouki, Mohamed Jemni, Talel Ladhari, Deployment of Solving Permutation Flow Shop Scheduling Problem on the Grid, Grid and Distributed Computing, Control and Automation, Springer Publisher, Vol 121, pp 95-104, 2010.